Flexible Score Aggregation

(Discussion Paper)

Paolo Ciaccia[1] and Davide Martinenghi[2]

[1]Università di Bologna, Italy, email: `paolo.ciaccia@unibo.it`
[2]Politecnico di Milano, Italy, email: `davide.martinenghi@polimi.it`

Abstract. Ranking objects according to different criteria is a central issue in many data-intensive applications. Yet, no existing solution deals with the case of partially specified score aggregation functions (e.g., a weighted sum with no precisely known weight values).
We address multi-source top-k queries with constraints (rather than precise values) on the weights. Our solution is instance optimal and provides increased flexibility with negligible overhead wrt classical top-k queries.

1 Introduction

Looking for information relevant to decide about the goodness of items of interest (hotels, restaurants, products, services, etc.) has become a common activity, supported by the so-called *multi-source top-k queries* [6]. For instance, the problem of choosing where to eat (possibly with constraints on price, food type, and location) may be addressed by *aggregating* different pieces of information from several sources. It is custom to assume that each such source provides a *ranked list* of results (from best to worst), each coming with a (local) score. Accordingly, objects can be either retrieved sequentially via a *sorted access* or by providing the object's identifier and obtaining the corresponding local score (*random access*). Then, for each object o in the dataset, a *monotone scoring function* f is used to aggregate the local scores yielding the (overall) score $f(o)$.

	a	h	j	d	f	b	e	i	c	g
R_1	9	8	8	7	7	6	6	5	4	4

	i	a	g	j	c	h	f	d	e	b
R_2	10	9	8	8	7	6	5	5	5	4

	c	f	e	i	d	a	b	g	h	j
R_3	9	8	7	7	6	5	5	5	4	3

Fig. 1: A set of ten restaurants ranked by three sources.

Example 1. Consider the scenario depicted in Figure 1, where ten restaurants, $\texttt{a}, \ldots, \texttt{j}$, are ordered by $d = 3$ rankings. Assume that the overall score is a weighted sum of the local scores, i.e., $f(o) = \sum_{i=1}^{d} w_i \cdot o[i]$, where $o[i]$ is the score assigned by R_i to object o (e.g., $\texttt{c}[2] = 7$), and that the $k = 2$ best restaurants are sought. In particular, using the *weight vector* $W_1 = (1/3, 1/3, 1/3)$, the top-2 restaurants are a and i, with $f(\texttt{a}) \approx 7.67$ and $f(\texttt{i}) \approx 7.33$. A different choice of weights, say $W_2 = (0.2, 0.3, 0.5)$, results in i and c being the top-2 restaurants.

Choosing precise weights in the scoring function is admittedly difficult, and might be the result of learning from users, e.g., via crowdsourcing [4]. A more viable alternative is to specify weights only partially through a set of *constraints*. With reference to Example 1, instead of having fixed weights, it might be interesting to "explore" all the alternatives arising when weights vary around W_2, by, say, ± 0.1. Then, only `a`, `c`, `f`, and `i` can be the possible (top-2) answers.

We study the problem of answering multi-source top-k queries with partially specified weights. This leads to a *family* $\mathcal{F}$ of scoring functions for ranking objects. Existing solutions to the problem with a single scoring function are *Fagin's algorithm* (`FA`) [6] and the *threshold algorithm* (`TA`) [7]. The latter was shown to perform better and to achieve optimal I/O performance. Our proposal, `FSA`, solves the general case and reduces to `TA` for classical top-k queries (i.e, when $\mathcal{F} = \{f\}$) and is *instance optimal for any* $\mathcal{F}$. When no constraints are present, `FSA` behaves like `FA` and returns the so-called *k-skyband* [9], the case $k = 1$ being the *skyline* of the dataset [1].

This paper summarizes the main contributions presented in [3], namely 1) a provably correct and instance optimal algorithm, `FSA`, for answering multi-source top-k queries, 2) several optimization opportunities for improving the execution time of `FSA`, 3) experimental evidence that `FSA` enjoys the flexibility of partially specified weights while incurring little overhead wrt top-k queries.

2 Problem statement

Consider a dataset $\mathcal{D} = \{o_1, \ldots, o_N\}$ where each object $o \in \mathcal{D}$ has $d \geq 2$ *partial scores*, $(o[1], \ldots, o[d])$. For any scoring function f and dataset $\mathcal{D}$, a *top-k* query on $\mathcal{D}$ using f returns a set of k objects with the highest scores according to f (ties arbitrarily broken). Given objects o and o', if $o[i] \geq o'[i]$ holds for all $i = 1, \ldots, d$, at least once strictly, then o *dominates* o', written $o \succ o'$. Notice that $o \succ o'$ implies $f(o) \geq f(o')$ $\forall f \in \mathcal{M}$ ($\mathcal{M}$ being the set of all monotone scoring functions). The *skyline* of $\mathcal{D}$ is the set of undominated objects in $\mathcal{D}$:

$$\textsc{Sky}(\mathcal{D}) = \{o \in \mathcal{D} \mid \nexists o' \in \mathcal{D}.\ o' \succ o\}, \tag{1}$$

but can also be viewed as the set of objects that can be top-1 [8], i.e.:

$$o \in \textsc{Sky}(\mathcal{D}) \iff \exists f \in \mathcal{M}.\ \forall o' \in \mathcal{D}.\ o' \neq o \to f(o) > f(o').$$

The *k-skyband* of $\mathcal{D}$, written $\textsc{Sky}_k(\mathcal{D})$, is the set of all objects in $\mathcal{D}$ that are dominated by less than k objects [9], and thus includes all possible top-k sets.

As a generalization of dominance for a family $\mathcal{F}$ of monotone scoring functions ([2]), we say that object o *$\mathcal{F}$-dominates* object o', $o \neq o'$, denoted by $o \succ_{\mathcal{F}} o'$, iff $\forall f \in \mathcal{F}.\ f(o) \geq f(o')$ and $\exists f \in \mathcal{F}.\ f(o) > f(o')$.

We define the *non-dominated restricted skyband* of $\mathcal{D}$ wrt $\mathcal{F}$, denoted $\textsc{nd}_k(\mathcal{D}; \mathcal{F})$, as the set of objects in $\mathcal{D}$ that are $\mathcal{F}$-dominated by less than k objects:

$$\textsc{nd}_k(\mathcal{D}; \mathcal{F}) = \{o \in \mathcal{D} \mid \nexists o_1, \ldots, o_k \in \mathcal{D}.\ \ o_1 \succ_{\mathcal{F}} o \wedge \ldots \wedge o_k \succ_{\mathcal{F}} o\} \tag{2}$$

Clearly, $\textsc{nd}_k(\mathcal{D}; \mathcal{F}) \subseteq \textsc{Sky}_k(\mathcal{D})$. When $\mathcal{F}$ reduces to a single function f, then $\textsc{nd}_k(\mathcal{D}, \{f\})$ includes *all* possible top-k results for f (whereas, in case of ties, top-k queries may nondeterministically discard tuples). From (2) it is evident that if $o \notin \textsc{nd}_k(\mathcal{D}; \mathcal{F})$, then *no scoring function $f \in \mathcal{F}$ can make o top-k*. We now address the problem of efficiently computing $\textsc{nd}_k(\mathcal{D}; \mathcal{F})$ in a multi-source scenario, for any given dataset $\mathcal{D}$ and set of monotone scoring functions $\mathcal{F}$.

3 Flexible score aggregation with `FSA`

Fagin's algorithm `FA` was the first to address multi-source top-k queries for arbitrary scoring functions. It consists of three phases: 1) sorted accesses are executed; 2) random accesses are executed on all objects met in phase 1; 3) scores of all fetched objects are computed using a scoring function f and the top-k result determined. The *stopping condition* for phase 1 is to have seen at least k objects on *all* the d rankings. With the scenario depicted in Figure 1, `FA` would stop at *depth* 7 (after 7 sorted accesses per ranking), after seeing `a` and `f` on all rankings. The `FA` algorithm can be adapted to compute $\textsc{Sky}_k(\mathcal{D})$, since it stops phase 1 independently of f. Therefore, the set S of objects seen when the algorithm stops includes the top-k objects according to *any* possible monotonic scoring function (and, thus, $\textsc{Sky}_k(\mathcal{D})$, which, in turn, includes $\textsc{nd}_k(\mathcal{D};\mathcal{F})$).

The `TA` algorithm uses a *threshold value* $f(\tau)$ computed by applying the specific scoring function f to the tuple τ of last scores seen by sorted access on each ranking, which we here call the *threshold point.* A further difference with respect to `FA` is that random accesses are not delayed to a separate phase: after performing d sorted accesses (one per ranking), all the necessary random accesses are executed. The algorithm stops when at least k seen objects have a global score no worse than $f(\tau)$. Thus, for each object o in the result, $f(o) \geq f(\tau)$ holds. In Figure 1, with $k = 2$ and weights $W_2 = (0.2, 0.3, 0.5)$ `TA` would stop at depth 4, where $f(\tau) = 7.3$, since $f(\mathtt{i}) = 7.5$ and $f(\mathtt{c}) = 7.4$.

Being tailored to a specific f, the stopping criterion of `TA` is (much) more efficient than the one used by `FA`, but now, when `TA` stops, S is only guaranteed to contain the top-k objects according to f, but not necessarily according to other scoring functions.

Let us say that *o weakly $\mathcal{F}$-dominates o'*, $o \neq o'$ iff $\forall f \in \mathcal{F}.\ f(o) \geq f(o')$. Then, we can unify `FA` and `TA` according to the following observation.

Observation 1 *`FA` stops executing sorted accesses when k seen objects $o_1, \ldots, o_k$* weakly $\mathcal{M}$-dominate τ. *`TA` stops when k seen objects* weakly $\{f\}$-dominate τ.

Our proposal, `FSA`, applies the notion of $\mathcal{F}$-dominance wrt τ, which includes ($\mathcal{M}$-)dominance and $\{f\}$-dominance as special cases. The extraction performed by `FSA` stops as soon as k objects are seen that $\mathcal{F}$-dominate τ, as all unseen objects are also necessarily $\mathcal{F}$-dominated by these k objects. Notice that `FSA` uses $\mathcal{F}$-dominance rather than weak $\mathcal{F}$-dominance for not discarding ties from $\textsc{nd}_k(\mathcal{D};\mathcal{F})$. The pseudocode of `FSA` is presented in Figure 1.

Note that the descent performed by `FSA` alternates a single sorted access on the i-th ranking with $d - 1$ random accesses on the other rankings. This differs from the strategy adopted by `TA`, which, at each depth, first performs d sorted accesses (one per ranking), thereby retrieving up to d unseen objects, and then completes the scores for these objects with up to $d \cdot (d - 1)$ random accesses. However, both algorithms stop at the same depth. Also note that `FA` performs all random accesses at the end; however, such an approach would not be possible for `TA` or `FSA`, since $\{f\}$-dominance and $\mathcal{F}$-dominance generally require knowing all the scores of the objects being compared.

Algorithm 1: FSA algorithmic pattern for computing ND_k.

```
Input: Rankings $R_1, \ldots, R_d$ for $\mathcal{D}$, family of scoring functions $\mathcal{F}$, integer $k > 0$.
Output: $\text{ND}_k(\mathcal{D}; \mathcal{F})$.
 1. let $S := \emptyset$ // Seen objects
 2. do
 3.   for $i$ in $1 \ldots d$
 4.     let $\langle o, \sigma \rangle :=$ sortedAccess($R_i$) // Do a sorted access in each ranking $R_i$
 5.     $\tau[i] := \sigma$ // Update score for threshold $\tau$
 6.     if $o \notin S$ then // Object $o$ is new
 7.       $o[i] := \sigma$ // Save score for $o$
 8.       for $j$ in $1 \ldots d$ // Extract all other scores for $o$ via random access
 9.         if $j \neq i$ then $o[j] :=$ randomAccess($R_j, o$)
10.       if $\nexists o_1, \ldots, o_k \in S.\ \ o_1 \succ_{\mathcal{F}} o \wedge \ldots \wedge o_k \succ_{\mathcal{F}} o$ then handle($o$)
11. while $\nexists o_1, \ldots, o_k \in S.\ \ o_1 \succ_{\mathcal{F}} \tau \wedge \ldots \wedge o_k \succ_{\mathcal{F}} \tau$ // Stop if $k$ objects $\succ_{\mathcal{F}} \tau$
12. clean() // Remove from $S$ all objects $\mathcal{F}$-dominated by $\geq k$ others
13. return $S$
```

Let $I = \langle \mathcal{D}, \mathcal{F}, k \rangle$ indicate an instance of the problem of computing $\text{ND}_k(\mathcal{D}; \mathcal{F})$ wrt a set of monotone scoring functions $\mathcal{F}$ and let $\mathbf{I}$ indicate the class of all such problems. An algorithm A is *correct* (for the computation of ND_k) if, for each instance $I = \langle \mathcal{D}, \mathcal{F}, k \rangle \in \mathbf{I}$, A returns $\text{ND}_k(\mathcal{D}; \mathcal{F})$.

Theorem 1. *FSA is correct for the computation of* ND_k.

Performance in top-k contexts is commonly characterized by the number of objects accessed by sorted access by an algorithm A on an instance I, which we indicate as $\texttt{sumDepths}(A, I)$. Algorithm A is *instance optimal* if there exist constants c_1 and c_2 such that $\texttt{sumDepths}(A, I) \leq c_1 \cdot \texttt{sumDepths}(A', I) + c_2$ for every correct algorithm A' and instance $I \in \mathbf{I}$.

Theorem 2. *FSA is instance optimal.*

4 Algorithmic variants

We now discuss efficient implementations of FSA and focus on the case in which each scoring function is of the form $f(o) = \sum_{i=1}^{d} w_i g_i(o[i])$, where the w_i's are *weights* subject to a set $\mathcal{C}$ of linear constraints and the g_i's are monotone. This form includes, e.g., weighted linear sums and weighted L_p norms. The constraints may be of several kinds (see [5] for an overview), such as

1. *weak rankings*: $w_1 \geq w_2 \geq \ldots \geq w_d$ (relative importance of attributes);
2. *ratio bounds*: $\bar{w}_i(1-\varepsilon) \leq w_i \leq \bar{w}_i(1+\varepsilon)$, $1 \leq i \leq d$ (spread around a weight).

Without loss of generality, we consider the space of normalized weight vectors $\mathcal{W} \subseteq [0,1]^d$, such that, for each $W = (w_1, \ldots, w_d) \in \mathcal{W}$, we have $\sum_{i=1}^{d} w_i = 1$; $\mathcal{W}(\mathcal{C})$ denotes the subset of $\mathcal{W}$ of weight vectors satisfying constraints $\mathcal{C}$.

$\mathcal{F}$-dominance test. There are two main approaches to efficiently check $\mathcal{F}$-dominance [2]. One approach requires solving a linear programming problem encoding the notion of $\mathcal{F}$-dominance. This may be expensive, since every $\mathcal{F}$-dominance test requires solving a different LP problem. A better approach, which we choose for `FSA`, is based on the *$\mathcal{F}$-dominance region* of an object o, i.e., the set of all points that o $\mathcal{F}$-dominates. Such a region needs to be computed only once for all the tests in which o is the candidate $\mathcal{F}$-dominant object. Testing membership to the $\mathcal{F}$-dominance region is done by *i)* computing the vertices of the convex polytope $\mathcal{W}(\mathcal{C})$, and *ii)* checking the following inequalities:

$$\sum_{i=1}^{d} w_i^{(\ell)} g_i(o[i]) \geq \sum_{i=1}^{d} w_i^{(\ell)} g_i(o'[i]), \quad \ell \in \{1, \ldots, q\}, \tag{3}$$

where each $W^{(\ell)} = (w_1^{(\ell)}, \ldots, w_d^{(\ell)})$, for $1 \leq \ell \leq q$, is a vertex of $\mathcal{W}(\mathcal{C})$, and at least one inequality is strict. Computing vertices may be expensive, but needs to be done only once per dataset.

Memoization of the vertex scores. We call the lhs of (3) the *vertex score* of o for $W^{(\ell)}$, denoted by $v_\ell(o)$ (similarly for o' in the rhs). Each vertex score, once computed, is remembered for subsequent tests (3) involving the same object. We adopt this technique for `FSA` and call it *memoization.*

Sorting. In `FSA`, we keep the seen objects topologically sorted wrt the $\mathcal{F}$-dominance relation, so that, in a sequence $o_1, \ldots, o_n$, no object o_i can be $\mathcal{F}$-dominated by an object o_j if $j > i$. We do so by sorting according to weights in the centroid of $\mathcal{W}(\mathcal{C})$. Then, when checking whether an object o should be added to the set S of seen objects (line 10 of Algorithm 1), we only look for k objects $\mathcal{F}$-dominating o among those preceding o in the topological sort.

Pruning. In `FSA`, object removal can occur within either `handle` or `clean`. With *lazy pruning*, we do not remove objects in `handle` and wait to do it when `clean` is called. With *eager pruning*, when an object o is inserted in S, `handle` also removes all objects in S that are $\mathcal{F}$-dominated by o and were already $\mathcal{F}$-dominated by $k-1$ other objects; `clean` does nothing.

									depth = 3		depth = 4		depth = 5	
ℓ	v_ℓ(a)	v_ℓ(i)	v_ℓ(c)	v_ℓ(h)	v_ℓ(f)	v_ℓ(j)	v_ℓ(g)	v_ℓ(e)	v_ℓ(d)	$v_\ell(\tau_3)$	V_ℓ	$v_\ell(\tau_4)$	V_ℓ	$v_\ell(\tau_5)$	V_ℓ
1	6.6	7.7	7.9	5.0	7.0	5.0	5.8	6.3	5.8	7.4	[f, a]	7.3	[f, a]	6.4	[]
2	7.0	8.0	7.7	5.2	6.7	5.5	6.1	6.1	5.7	7.5	[a, f]	7.4	[a, f]	6.5	[]
3	7.4	7.8	7.2	5.6	6.6	6.0	6.0	6.0	5.8	7.6	[a, c, f]	7.4	[c, f]	6.6	[]
4	7.4	7.3	6.9	5.8	6.8	6.0	5.6	6.1	6.0	7.6	[a, i, c, f]	7.3	[c, f]	6.6	[]
5	6.6	7.2	7.6	5.2	7.2	5.0	5.4	6.4	6.0	7.4	[i, f, a]	7.2	[a]	6.4	[]
6	7.0	7.0	7.1	5.6	7.1	5.5	5.3	6.3	6.1	7.5	[c, f, a, i]	7.2	[c, f, a, i]	6.5	[]

Table 1: Vertex scores for Examples 1 and 2, along with threshold point's vertex scores and vertex lists at depth 3, 4, and 5.

Vertex lists. We observe that τ is $\mathcal{F}$-dominated by an object o iff all vertex scores of o are no less than the corresponding vertex scores of τ, and at least one is greater. We then maintain, for each vertex $W^{(\ell)}$, a list V_ℓ of seen objects whose vertex score is still lower than τ's vertex score $v_\ell(\tau)$. An object o is added to V_ℓ within the call to `handle`(o) and, in V_ℓ, the objects are kept sorted by vertex score. Additionally, for each object o, we maintain a mask of q bits $B_1(o), \ldots, B_q(o)$ such that $B_\ell(o) = 1$ iff $v_\ell(o) \geq v_\ell(\tau)$, plus one extra bit $B_{>}(o)$

that is set to 1 iff at least one inequality holds strictly. Analogously, we also maintain a list $V_>$ of those objects o such that $B_>(o) = 0$. In this way, when checking which objects $\mathcal{F}$-dominate τ, it suffices to consider, for each $\ell \in \{1, \ldots, q\}$, only the objects in $V_\ell \cup V_>$ and only if $v_\ell(\tau)$ has changed since the last check.

Example 2. Consider again the scenario of Example 1, with $k = 2$ and weight vector $(0.2, 0.3, 0.5)$ ± 0.1 on each weight. The vertices of $\mathcal{W}(\mathcal{C})$ are $W^{(1)} = (0.1, 0.3, 0.6)$, $W^{(2)} = (0.1, 0.4, 0.5)$, $W^{(3)} = (0.2, 0.4, 0.4)$, $W^{(4)} = (0.3, 0.3, 0.4)$, $W^{(5)} = (0.2, 0.2, 0.6)$, $W^{(6)} = (0.3, 0.2, 0.5)$, with centroid $(0.2, 0.3, 0.5)$. At depth 3, the seen objects are `a`, `i`, `c`, `f` (which will all appear in the result) and `h`, `j`, `g`, `e`, which are all $\mathcal{F}$-dominated by both `i` and `c` and thus not retained. The objects are inserted sortedly in S according to the centroid, with the order: `i`, `c`, `a`, `f`. Table 1 shows the vertex scores, orderly computed as $\sum_{i=1}^{d} w_i^{(\ell)} o[i]$ (`b` is not even accessed). To see that, e.g., `i` $\succ_\mathcal{F}$ `h` we observe that $v_\ell(\texttt{i}) \geq v_\ell(\texttt{h})$, for $1 \leq \ell \leq q$ (and $v_1(\texttt{i}) > v_1(\texttt{h})$). At this point, the seen objects S already coincide with $\textsc{nd}_2$, but `FSA` cannot stop yet. The threshold point is $\tau_3 = (8, 8, 7)$; its vertex scores are shown in Table 1, along with the corresponding vertex lists (of objects that still have a vertex score lower than τ_3's). The bit masks $B_>(o)B_1(o) \ldots B_6(o)$ are computed accordingly; e.g., we have 1110010 for `c` since `c` has a worse vertex score than τ_3 on vertices $W^{(3)}$, $W^{(4)}$, and $W^{(6)}$, and thus is present in vertex lists V_3, V_4, and V_6. At depth 4, $\tau_4 = (7, 8, 7)$, but still no object $\mathcal{F}$-dominates it. Finally, at depth 5, $\tau_5 = (7, 7, 6)$, which is now $\mathcal{F}$-dominated by all of `i`, `c`, `a`, `f`, and then `FSA` can stop. The subsequent `clean` procedure does not eliminate anything, since none of the remaining objects is $\mathcal{F}$-dominated by $k = 2$ other objects, thus $\textsc{nd}_2(\mathcal{D}; \mathcal{F}) = \{\texttt{i}, \texttt{c}, \texttt{a}, \texttt{f}\}$.

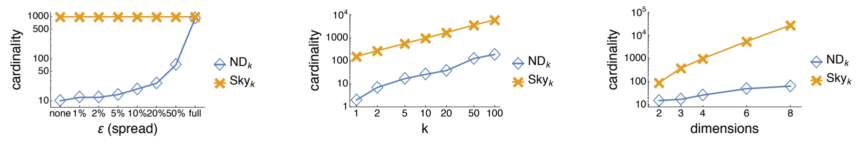
5 Experiments

In this section we evaluate the performance of our implementations of `FSA`. In particular, we quantify the overhead incurred by the increased flexibility of `FSA` when compared to the computation of plain k-skybands and top-k queries. For our experiments, we consider three kinds of data distribution (anticorrelated, uniform, and real)[1] and scoring functions taken from weighted L_1 norms with ratio bounds constraints applied on the weights $(1/d(1 - \varepsilon) \leq w_i \leq 1/d(1 + \varepsilon))$. We report on both effectiveness and efficiency in different scenarios varying by number of rankings d, spread ε, and k, with defaults 4, 20%, and 10, resp.

In order to evaluate the effectiveness of our approach, we compare it to k-skybands, which also retrieve non(-$\mathcal{F}$)-dominated objects but in a way that is agnostic with respect to the set of scoring functions $\mathcal{F}$. The introduction of $\mathcal{F}$ drastically helps to reduce the result set. Figure 2 shows this effect for `NBA`. In Figure 2a, we vary the spread parameter ε used in ratio bounds constraints and keep default values for all other parameters. As can be seen, the result sets returned by $\textsc{nd}_k$ and $\textsc{Sky}_k$ coincide (985 objects) only when the spread is so high

[1] `NBA`: 190862 tuples reporting measures of basketball players' performance, available at `www.quantifan.com`; `ANT` and `UNI` have $100K$ tuples.

that each weight can vary in the entire $[0, 1]$ interval ("full"). For all other values of ε, the result set of ND_k is much smaller (from 71 objects when $\varepsilon = 50\%$, to only 12 for $\varepsilon = 1\%$). Note that, when $\varepsilon = 0\%$ ("none"), all weights equal $1/d$ exactly and, thus, there is only one scoring function in $\mathcal{F}$; in that case, the cardinality of the result set is exactly $k = 10$. Figure 2b also reports the cardinality of the result sets, but for different values of k (defaults otherwise). The result set returned by Sky_k is always almost two orders of magnitude larger than that of ND_k, the ratio becoming larger as k increases (from 157/2 when $k = 1$ to 6076/191 when $k = 100$). Finally, Figure 2c shows this when the number of rankings d varies (defaults otherwise). As d increases, the ratio between the cardinalities of the results increases (from 89/15 when $d = 2$ to 27403/63 when $d = 8$). Similar trends can be observed for the `UNI` and `ANT` datasets (omitted in the interest of space). Overall, this shows that, thanks to $\mathcal{F}$, ND_k is able to keep the result set within a reasonable cardinality, even for larger or more challenging problem instances, while the cardinality of Sky_k soon diverges.



(a) Spread ε varies.

(b) k varies.

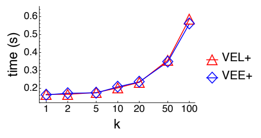
(c) Number of rankings d varies.

Fig. 2: Cardinality of results for the `NBA` dataset. Varying parameter: ε in (a), k in (b), d in (c).

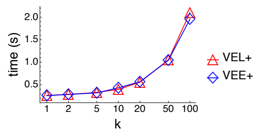
We consider two variants of the `FSA` pattern: `VEL+` and `VEE+`, using lazy and, resp., eager pruning, and test them in terms of execution time.

Effect of number of rankings d. Both variants exhibit sub-second execution times in default conditions. The "vertex lists" optimization grants savings of up to one order of magnitude, especially for more complex problem instances. Indeed, for $d > 4$ the number q of vertices increases dramatically, from $q = 6$ with $d = 4$ to $q = 70$ with $d = 8$. Yet, our optimization allows skipping many (already verified) vertex score comparisons between objects and the threshold point, thus leading to a linear behavior as d varies.

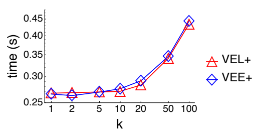
Effect of k and spread ε. Figure 3 shows execution times as k and ε vary in all datasets. Clearly, in all scenarios, the problem becomes more challenging as k increases and as ε increases, since, in both cases, more objects need to be extracted and compared. `VEL+` and `VEE+` show similar performance, with `VEE+` slightly prevailing on more challenging instances (e.g., when $k = 100$ or $\varepsilon =$"full"). In such cases, many objects are kept by a lazy strategy and thus the call to the `clean` procedure weighs more heavily than the eager pruning of objects. In the other cases, the extra cost of pruning objects early is balanced with the reduced cost of calling `clean`. We also observe that, in most scenarios, `VEE+` and `VEL+` incur sub-second execution times, and only exceed $2s$ when $\varepsilon =$"full" (a case in which the result coincides with the k-skyband) in the `ANT` dataset.



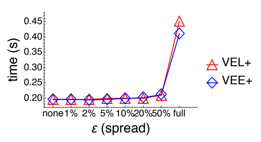
(a) `UNI`: k varies.



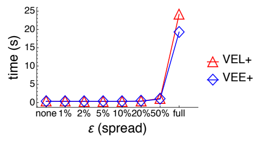
(b) `ANT`: k varies.



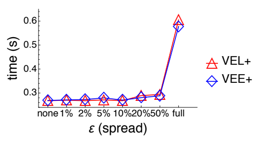
(c) `NBA`: k varies.



(d) `UNI`: spread ε varies.



(e) `ANT`: spread ε varies.



(f) `NBA`: spread ε varies.

Fig. 3: CPU times for computing ND_k. Datasets: `UNI` in (a),(d); `ANT` in (b),(e); `NBA` in (c),(f).

6 Conclusions

We have presented a framework for multi-source top-k queries defined by scoring functions with partially specified weights. This provides the flexibility required in many application scenarios (e.g., online services, crowdsourcing, etc.). Our solution, `FSA`, is a provably instance optimal algorithmic pattern that encompasses both top-k queries (attaining comparable performance) and k-skyband queries.

References

1. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
2. P. Ciaccia and D. Martinenghi. Reconciling skyline and ranking queries. *PVLDB*, 10(11):1454–1465, 2017.
3. P. Ciaccia and D. Martinenghi. FA + TA < FSA: Flexible score aggregation. In *CIKM 2018*, pages 57–66, 2018.
4. E. Ciceri et al. Crowdsourcing for top-k query processing over uncertain data. *TKDE*, 28(1):41–53, 2016.
5. Y. S. Eum et al. Establishing dominance and potential optimality in multi-criteria analysis with imprecise weight and value. *Computers & Op. Res.*, 28:397–409, 2001.
6. R. Fagin. Combining fuzzy information from multiple systems. In *PODS*, pages 216–226, 1996.
7. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
8. N. Meneghetti et al. Output-sensitive evaluation of prioritized skyline queries. In *SIGMOD*, pages 1955–1967, 2015.
9. D. Papadias et al. Progressive skyline computation in database systems. *TODS*, 30(1):41–82, 2005.